Real-Time Glints Rendering with Prefiltered Discrete Stochastic Microfacets

Beibei Wang, Hong Deng, Nicolas Holzschuch

Volume 38 (2019), Number 7 pp. 1–9 COMPUTER GRAPHICS *forum*

Real-Time Glints Rendering with Prefiltered Discrete Stochastic Microfacets

Beibei Wang[1,2], Hong Deng[1,2],Nicolas Holzschuch[3]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology
[2]Key Lab of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education
[3]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK
beibei.wang@njust.edu.cn, hong.deng.bot@njust.edu.cn, nicolas.holzschuch@inria.fr

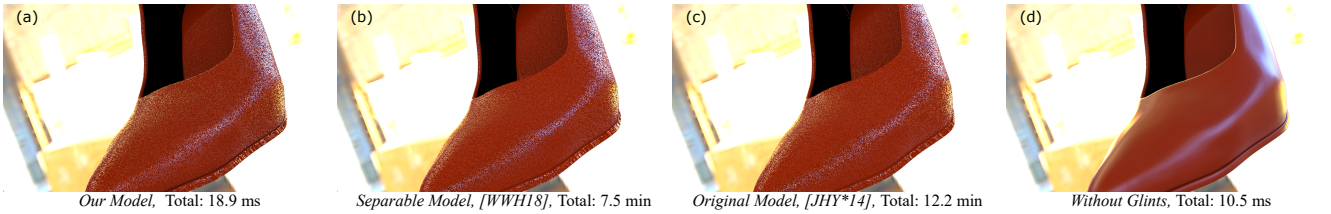


Figure 1: *Our algorithm (a), compared to the original Discrete Stochastic Microfacets model [JHY*14] (c) and the Separable Discrete Stochastic Microfacets model [WWH18] (b). Our algorithm produces similar pictures, with only 8.4 ms for glint evaluation and 18.9 ms for entire rendering. Glint Material: roughness 0.1, glint count N: 1M, search range γ: 6.0.*

Abstract

Many real-life materials have a sparkling appearance. Examples include metallic paints, sparkling fabrics, snow. Simulating these sparkles is important for realistic rendering but expensive. As sparkles come from small shiny particles reflecting light into a specific direction, they are very challenging for illumination simulation. Existing approaches use a 4-dimensional hierarchy, searching for light-reflecting particles simultaneously in space and direction. The approach is accurate, but extremely expensive. A separable model is much faster, but still not suitable for real-time applications.

The performance problem is even worse when illumination comes from environment maps, as they require either a large sample count per pixel or prefiltering. Prefiltering is incompatible with the existing sparkle models, due to the discrete multi-scale representation. In this paper, we present a GPU friendly, prefiltered model for real-time simulation of sparkles and glints. Our method simulates glints under both environment maps and point light sources in real-time, with an added cost of just 10 ms per frame with full high definition resolution. Editing material properties requires extra computations but is still real-time, with an added cost of 10 ms per frame.

Keywords: Rendering, surface microstructure, glints, real-time, prefiltered

CCS Concepts

• **Computing methodologies** → **Ray tracing; Reflectance modeling;**

1. Introduction

Many materials in real-life exhibit sparkles. Examples include but are not limited to snow, car paints, lipsticks, fabrics and sand. These sparkling effects are essential to the appearance of the material. Photorealistic simulation requires simulating these effects, whether it is for real-time rendering or for offline rendering. However, glints and sparkles are difficult to simulate, precisely because of their features: they are both spatially sensitive, as glints come from small particles with random distribution, and directionally sensitive, as glints provide bright reflection for a specific direction. Without the proper model and accurate sampling, simulating glints will result in noise or aliasing.

Jakob et al. [JHY*14] introduced a discrete stochastic microfacet model for glints. This model represents glints as a small specular surface, as in the microfacet model [CT82] [WMLT07], but with a discrete distribution instead of a continuous surface. Glints are rep-

resented as small specular particles, organized in a 4 dimensional hierarchy (space and angle). During rendering, they explore this 4-dimensional hierarchy to count the number of particles that fall within the pixel footprint, with a surface normal that is close to the half-vector between incoming and outgoing directions. These particles are used to compute the distribution function and the reflectance. This model is physically based but the 4 dimensional hierarchy traversal makes it slow. Atanasov and Koylazov [AK16a] replaced the 4 dimensional traversal with a 2 dimensional traversal in the texture space and explicitly generated the flake normals at the end of the traversal, resulting in higher performance. Wang et al. [WWH18] introduced a separable version, replacing the 4-dimension traversal with two separate 2D traversals resulting in faster simulation. The parameter separation also makes it easier to prefilter indirect illumination.

These models are purely procedural, without the need to store or compute textures. They are also physically-based and temporally coherent. The separable model by Wang et al. [WWH18] is faster than the original model, but still too slow for interactive or real-time rendering. These models were designed for offline rendering, where we evaluate material reflectance for a single direction. With environment map lighting, these models result in obvious noise in the picture. Reducing this noise by increasing the sample count would only further degrade the performance, making the models unsuitable for interactive applications. For real-time rendering with illumination from an environment map, we generally prefilter the environment map [KVHS00]. The multi-scale nature of the glint model [JHY*14] makes it incompatible with prefiltering, as the filter kernel should depend on the pixel footprint, which is only known at run-time. The separable model by Wang et al. [WWH18] is theoretically compatible with prefiltering. In practice, it is not straightforward to combine it with prefiltering, for two reasons: first, averaging by prefiltering can cancel the random nature of glittery appearance; second, even combined with prefiltering, the rendering cost of Wang et al. [WWH18] is too expensive for real-time rendering. We address both issues in this paper.

Our paper presents a new model for glints, resulting in noise-free glint rendering under environment map illumination in less than 30 ms per frame for the entire rendering with full high definition (FHD) resolution, of which just about 10 ms are for the glint evaluation. Our model also allows interactive editing of glint properties, at an extra cost of about 10 ms per frame. Specifically, our contributions are:

- An approximated separate radiance convolution model for microfacet multi-scale BRDFs, to make it compatible with prefiltering.
- A complete prefiltering method for the stochastic discrete microfacet model, based on prefiltering for glint probability and radiance term.
- A three-scale directional filtering method for further acceleration.

We review previous work on rendering glints and sparkles in the next section. We then present the original and separable discrete stochastic microfacet model in Section 3. We describe our own algorithm in Section 4. In Section 6, we compare our method with previous works and reference solutions. We conclude in Section 7.

2. Previous Work

2.1. Models for glints and scratches

Glints and scratches are surface details. The first issue for rendering them is to model them. Glints can be represented as a procedural model, without storage, using a stochastic approach. Scratches require more information, and are usually expressed using normal maps. The latter is more generic: approaches that model scratches can also be used to represent glints. The procedural approach, however, has a very low memory cost that makes it more practical for GPU implementations.

Jakob et al. [JHY*14] introduced a stochastic model where glints are small specular particles whose position and orientation are given by the microfacet model [CT82] [WMLT07], but with a discrete distribution instead of a continuous surface. This model was designed for offline rendering, and produces high-quality, physically based glints. Atanasov et al. [AK16b] extended this work using a different microfacet distribution. Wang et al. [WWH18] extended Jakob et al. [JHY*14] model by making the model separable and filterable, to get glittery effects more efficiently. Both extensions are also designed for offline rendering.

Yan et al. [YHJ*14] introduced a model for scratches based on a normal map representation. Their model computes accurate reflectance values using a hierarchical search to locate normals that are close to the half-vector. Yan et al. [YHMR16] improves on this approach by approximating a 4D distribution as a mixture of Gaussian elements. Both approaches were designed for offline rendering; the memory cost associated with normal map storage makes it difficult to port them to the GPU.

Gamboa et al. [GGN18] proposed a filtering approach for glints and scratches with large environmental lights, by computing the integral of filtered high-frequency reflectance over large lights with angularly varying emission, resulting in much faster rendering without quality degradation. Although it's much faster than prior works, it is still designed for offline rendering.

2.2. Approximate glint models

Separately, researchers have worked on approximate representations for glints. Shopf [Sho12] represented sparkles in the snow in real-time, using a jittered 3D grid. Bowles and Wang [BW15, WB16] introduced a procedural sparkle approach for snow; this technique has been used in production. Zirr and Kaplanyan [ZK16a] derived a stochastic bi-scale microfacet model to fit for real-time application. These four algorithms were designed for fast rendering and are not physically based. The glints caused by different light sources are approximated by different particle reflection cones which are computed from the material roughness and the light cone.

In comparison with previous work, our method is based on the stochastic model of Jakob et al. [JHY*14], but designed for real-time rendering. It is much faster than the models of Jakob et al. [JHY*14] or Wang et al. [WWH18], while providing results that are almost identical. It is slightly slower than the approximate model of Zirr and Kaplanyan [ZK16a], but provides results that are closer to the reference solution.

2.3. Prefiltering Environment maps

For illumination under environment map lighting, we usually prefilter the environment map during a preprocessing step, computing a convolution between the environment map and the BRDF. For an isotropic BRDF, this requires a 4-dimension representation. Kautz et al. [KVHS00] and McAllister et al. [MLH02] proposed approaches to compress this representation. Wang et al. [WRG*09] represented the BRDF as a sum of Spherical Gaussians for environment map prefiltering.

To avoid precomputation, Křivánek and Colbert [KC08] introduced filtered importance sampling, combining BRDF importance sampling and environment map prefiltering. The proposed method achieved real-time rendering of glossy objects under image-based illumination and allowed both dynamic lighting and BRDF. Their method requires an analytical formula for the BRDF, which is not available for glints using the discrete BRDF model [JHY*14].

In this work, we prefilter a multi-scale discrete microfacet model. We do not focus on representation compression, so we use a naive table representation. All existing compression methods, e.g. representation with Spherical Gaussians [WRG*09] can be used in combination with our approach.

3. Background

3.1. Discrete Stochastic Microfacet Model

The Discrete Stochastic Microfacet Model [JHY*14] represents glittery surfaces. The key idea is to start with the classical microfacet BRDF model, but with a finite extent in space and angle instead of dirac:

$$\hat{f}_r(A,\boldsymbol{i},\Omega_o) = \frac{1}{a(A)\sigma(\Omega_o)}\int_A\int_{\Omega_o} f_r(\boldsymbol{x},\boldsymbol{i},\boldsymbol{o})\,\mathrm{d}\boldsymbol{o}\,\mathrm{d}\boldsymbol{x}. \tag{1}$$

Where A is a finite area around point $\boldsymbol{x}$, Ω_o is a finite solid angle around outgoing direction $\boldsymbol{o}$, $a(A)$ is the surface area of A, $\sigma(\Omega_o)$ denotes the area of Ω_o on the unit sphere, and f_r is the usual microfacet BRDF [CT82] [WMLT07]:

$$f_r(\boldsymbol{x},\boldsymbol{i},\boldsymbol{o}) = \frac{F(\boldsymbol{i}\cdot\boldsymbol{h})D(\boldsymbol{x},\boldsymbol{h})G(\boldsymbol{i},\boldsymbol{o},\boldsymbol{h})}{4(\boldsymbol{i}\cdot\boldsymbol{n}_x)(\boldsymbol{o}\cdot\boldsymbol{n}_x)}. \tag{2}$$

Where F represents the Fresnel reflection coefficient, D is the microfacet normal distribution, G is the shadowing and masking term, and $\boldsymbol{n}_x$ is the shading normal.

They introduced an approximation format, using the cone central direction, and the change-of-measure term, which yields the final flake BRDF model:

$$\hat{f}_r(A,\boldsymbol{i},\Omega_o) = \frac{(\boldsymbol{i}\cdot\boldsymbol{h})F(\boldsymbol{i}\cdot\boldsymbol{h})\hat{D}(A,\Omega_h)G(\boldsymbol{i},\boldsymbol{o},\boldsymbol{h})}{a(A)\sigma(\Omega_o)(\boldsymbol{i}\cdot\boldsymbol{n}_x)(\boldsymbol{o}\cdot\boldsymbol{n}_x)}, \tag{3}$$

where $\hat{D}$ is a sum over the finite set of particles:

$$\hat{D}(A,\Omega_h) := \frac{1}{N}\sum_{k=1}^{N} \mathbf{1}_{\Omega_h}(\boldsymbol{h}^k)\mathbf{1}_A(\boldsymbol{x}^k). \tag{4}$$

For each particle k, if its position $\boldsymbol{x}^k$ and half-vector $\boldsymbol{h}^k$ both fall within the search domain in space and angle $A\times\Omega_h$, then the element in the sum is 1. In short, $\hat{D}$ is the fraction of particles that fall within this search domain. N is the total particle count. A usually corresponds to a pixel footprint, and Ω_h to a cone of directions around which particles can reflect light. The cone half-angle γ is a parameter of the model. In practical applications, it ranges from 1^o to 6^o.

3.2. Separable Model

Wang et al. [WWH18] replaced the particle count with a particle probability function, which can be carried out of the integral, allowing the decoupling between space and angle.

This function is called the *Directional Probability Function* (DPF), $P(\boldsymbol{i},\boldsymbol{o},\gamma)$, expressing the probability that a particle exists that reflects light incoming from direction $\boldsymbol{i}$ into a cone centered around direction $\boldsymbol{o}$ with half-angle γ.

They rewrote Equation 4 using this directional probability function:

$$\hat{D}(A,\boldsymbol{i},\boldsymbol{o}) = \left(\frac{1}{N}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)\right)P(\boldsymbol{i},\boldsymbol{o},\gamma) \tag{5}$$

The directional probability function $P(\boldsymbol{i},\boldsymbol{o},\gamma)$ is expressed as

$$P(\boldsymbol{i},\boldsymbol{o},\gamma) \approx \frac{1}{N}\sum_{k=1}^{N}\mathbf{1}_{\Omega_h}(\boldsymbol{h}^k) \tag{6}$$

The directional probability function P represents a *continuous* approximation of the flakes reflection. To reproduce the flakes *discrete* behaviour, they threshold it against a random variable of position, $\lambda(\boldsymbol{x}^k)$:

$$\hat{D}(A,\boldsymbol{i},\boldsymbol{o}) = \frac{1}{N}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)H\left(\lambda(\boldsymbol{x}^k)-P(\boldsymbol{i},\boldsymbol{o},\gamma)\right) \tag{7}$$

$$H(u) = \begin{cases} 1, & \text{if } u<0,\\ 0, & \text{otherwise.}\end{cases} \tag{8}$$

In practice, the DPF is precomputed by counting the particles for uniformly sampled directions similarly to [JHY*14], but considering only the angular domain and then normalizing by the total particle count.

During rendering, they do a spatial hierarchy traversal to find the particles located within the pixel footprint. They use a quadtree for the spatial hierarchy, representing the subdivision in texture space. For each subdivided node, they generate the particle count procedurally. For each particle found, they use the DPF to decide its acceptance.

3.3. Prefiltering for Environment Map

Under distant environment lighting, outgoing radiance is the convolution of incoming radiance and BRDF:

$$L_o(\boldsymbol{o}) = \int_\Omega L_i(\boldsymbol{i}')f_r(\boldsymbol{o},\boldsymbol{i}')\,\mathrm{d}\boldsymbol{i}' \tag{9}$$

To compute this integral in practice, we use importance-sampling with the BRDF model f_r to compute the sampling direction $\boldsymbol{i}'$. It takes a large number of samples to obtain a noise-free result.

For real-time computation, this integral is precomputed for dependent variables such as surface orientation and view direction.

This prefiltering approach does not work with the multi-scale discrete BRDF model [JHY*14] $\hat{f}_r$ inside the integral:

$$L_o(A,\boldsymbol{o}) = \int_\Omega L_i(\boldsymbol{i}')\hat{f}_r(A,\boldsymbol{o},\boldsymbol{i}')\,\mathrm{d}\boldsymbol{i}'. \tag{10}$$

The A in the Equation 10 represents the pixel footprint in the scene. It changes during run-time. The mechanic in which the glints computation work (Equation 8) makes it impossible to prefilter Equation 10 directly.

The separable model [WWH18] decouples spatial and directional terms, thus allowing directional prefiltering. However, it's not straightforward to use it directly, for two reasons: first, the random nature of glittery appearance can not be captured with the average prefiltering; second, the performance is not satisfying without our proposed three-scale model. In the next section, we present a practical prefiltering method and solve these two issues.

4. Real-time Glint Rendering with Discrete Stochastic Model

Figure 2 presents a summary of our algorithm for real-time rendering of glints, under illumination by environment maps. Our algorithm works by reformulating the Discrete Stochastic Microfacet model so it is compatible with prefiltering (Section 4.1). We preconvolve the environment map with our new glint microfacet model (Section 4.2). For faster rendering, we introduce a three-scale model, adapting computational intensity to glint importance on screen (Section 4.3). In section 4.4, we summarize the entire algorithm.

4.1. Prefilterable Multi-Scale Flake BRDF Model

With glints and sparkles, the amount of light being reflected in a specific direction depends on the number of glints in the pixel footprint, making the model incompatible with classical prefiltering. We rewrite the separable model [WWH18] (Equation 5) and show we can prefilter in the directional domain, then combine the result in the spatial domain during rendering.

In Equation 3, $\hat{D}$ is the multiple scale version from Equation 4. We group the non multi-scale terms in to a single term $M(\boldsymbol{i},\boldsymbol{o})$:

$$M(\boldsymbol{i},\boldsymbol{o}) = \frac{(\boldsymbol{i}\cdot\boldsymbol{h})F(\boldsymbol{i}\cdot\boldsymbol{h})G(\boldsymbol{i},\boldsymbol{o},\boldsymbol{h})}{(\boldsymbol{i}\cdot\boldsymbol{n}_x)(\boldsymbol{o}\cdot\boldsymbol{n}_x)}. \tag{11}$$

Now, we get a shorter version of the flake model:

$$\hat{f}_r(A,\boldsymbol{i},\Omega_o) = \hat{D}'(A,\boldsymbol{o},\boldsymbol{i})M(\boldsymbol{i},\boldsymbol{o}), \tag{12}$$

where $\hat{D}'$ represents $\hat{D}$ divided by the two terms $a(A)\sigma(\Omega_o)$.

Integrating the separable model from Equation 12 into the environment map integral (Equation 10) gives:

$$L_o(A,\boldsymbol{o}) = \int_\Omega L_i(\boldsymbol{i}')\hat{D}'(A,\boldsymbol{o},\boldsymbol{i}')M(\boldsymbol{i}',\boldsymbol{o})\,\mathrm{d}\boldsymbol{i}'. \tag{13}$$

We move Equation 5 into Equation 13:

$$L_o(A,\boldsymbol{o}) = \frac{1}{a(A)\sigma(\Omega_o)N}\int_\Omega L_i(\boldsymbol{i}')\left(\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)\right)P(\boldsymbol{i}',\boldsymbol{o},\gamma)M(\boldsymbol{i}',\boldsymbol{o})\,\mathrm{d}\boldsymbol{i}'. \tag{14}$$

The spatial flake counting does not depend on $\boldsymbol{i}'$, so we move it outside of the integral:

$$L_o(A,\boldsymbol{o}) = \frac{1}{a(A)\sigma(\Omega_o)N}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)\int_\Omega L_i(\boldsymbol{i}')P(\boldsymbol{i}',\boldsymbol{o},\gamma)M(\boldsymbol{i}',\boldsymbol{o})\,\mathrm{d}\boldsymbol{i}'. \tag{15}$$

We now have expressed illumination from the environment map in a form that we can preconvolve: the part under the integral is independent from the pixel footprint. $P(\boldsymbol{i}',\boldsymbol{o},\gamma)$ is the Directional Probability Function (DPF) (Equation 6) and $M(\boldsymbol{i}',\boldsymbol{o})$ is the the non multi-scale terms (Equation 11) in the multi-scale flakes BRDF model (Equation 3).

4.2. Glint Prefiltering with 4D Representation

We precompute the integral part of Equation 15 for combinations of normal direction $\boldsymbol{n}$ and outgoing direction $\boldsymbol{o}$. We call it $\hat{L}_o(\boldsymbol{n},\boldsymbol{o})$. For each pair of directions $(\boldsymbol{n},\boldsymbol{o})$, we generate multiple sample directions $\boldsymbol{i}_j$ using BRDF importance sampling and sum the contributions:

$$\hat{L}_o(\boldsymbol{n},\boldsymbol{o}) = \sum_{j=1}^{j=K}\frac{L_i(\boldsymbol{i}_j)P(\boldsymbol{i}_j,\boldsymbol{o},\gamma)M(\boldsymbol{i}_j,\boldsymbol{o})}{\mathrm{pdf}(\boldsymbol{i}_j)K}. \tag{16}$$

Where K is the sample count (set as 64 in our implementation) and $\mathrm{pdf}(\boldsymbol{i}_j)$ is the probability density function of direction $\boldsymbol{i}_j$.

We also compute an average value for P for each pair of directions $(\boldsymbol{n},\boldsymbol{o})$, using uniform sampling:

$$\hat{P}(\boldsymbol{n},\boldsymbol{o}) = \sum_{j=1}^{j=B}\frac{2\pi P(\boldsymbol{i}_j,\boldsymbol{o},\gamma)}{B}. \tag{17}$$

Where B is the sample count around the hemispherical space (set as 36×9) and $\boldsymbol{i}_j$ is the uniformly sampled direction.

In practice, we use a single lookup table to represent both. This table has 4 dimensions: 2 dimensions for the surface orientation $\boldsymbol{n}$ and 2 dimensions for the view direction $\boldsymbol{o}$. The number of samples on each dimension depends on the surface roughness. Table 2 shows the practical values we used.

At run-time, we replace the convolution of the BRDF and the environment map with the prefiltered radiance:

$$L_o(A,\boldsymbol{o}) = \frac{\hat{L}_o(\boldsymbol{n},\boldsymbol{o})}{a(A)\sigma(\Omega_o)N}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k), \tag{18}$$

where $\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)$ is the count of particles located in the pixel footprint, computed using a spatial traversal. To produce the glittery effect, we introduce the two average DPFs to Equation 18 and move one of them inside the sum:

$$L_o(A,\boldsymbol{o}) = \frac{\hat{L}_o(\boldsymbol{n},\boldsymbol{o})}{a(A)\sigma(\Omega_o)N\hat{P}(\boldsymbol{n},\boldsymbol{o})}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)\hat{P}(\boldsymbol{n},\boldsymbol{o}), \tag{19}$$

where the term inside the sum means the count of particles that satisfy both the spatial and angular conditions (DPF). By changing statistics probability for all the particles in the footprint to each

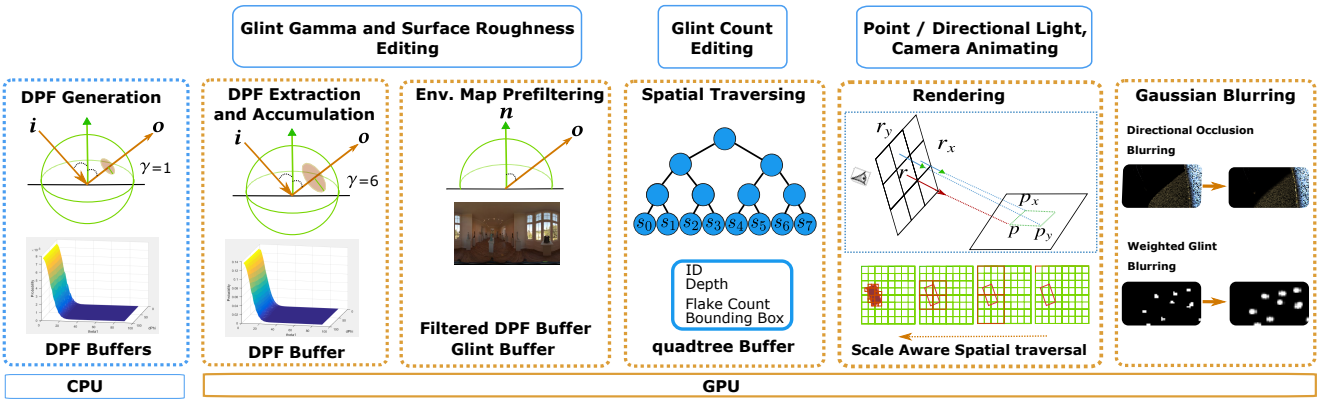


Figure 2: *Our algorithm: We first compute and store the probability for a particle to be reflected around the direction* $\boldsymbol{o}$ *with a tolerance of 1 degree. This probability is indexed by the incoming and outgoing directions* $\boldsymbol{i},\boldsymbol{o}$ *and stored in DPF buffers. This step runs on the CPU, all the others run on the GPU. Next (DPF Extraction and Accumulation), given the glint material properties (roughness* α *and search range* γ*), we extract the DPF buffer considering* α *and accumulate table values considering* γ*. Then we prefilter the environment map with the glint BRDF and the DPF for each sampled* $\boldsymbol{n},\boldsymbol{o}$*, resulting in a Filtered Glint Buffer and Filtered DPF Buffer. Then we build a spatial hierarchy for the imaginary flakes, resulting in a quadtree. In the* rendering *step, we compute the glints using the quadtrees and the buffers. We also compute directional occlusion, testing visibility along the direction corresponding to the specular reflection of viewing direction with respect to the surface normal. Finally, we blur directional occlusion with a large kernel, blur the computed glints with a smaller kernel, and multiply both buffers together to get the complete glint contribution. If material parameters stay constant, we only need to perform the rendering and filtering steps.*

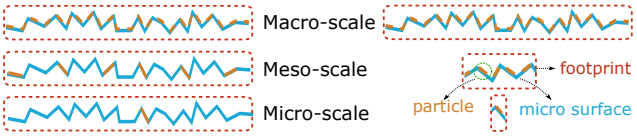


Figure 3: *The count of the particles determines the scale.*

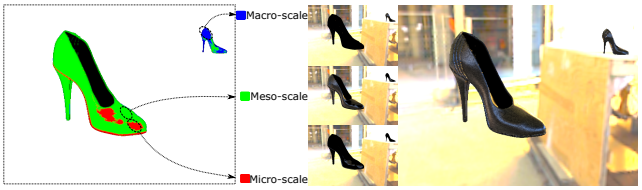


Figure 4: *Visualization of the three-scale (a) and each scale rendering result and their combination (b) on the Shoe Scene. Glint Material: roughness 0.1, N: 1M,* γ*: 6.0.*

particle, we get:

$$L_o(A,\boldsymbol{o}) = \frac{\hat{L}_o(\boldsymbol{n},\boldsymbol{o})}{a(A)\sigma(\Omega_o)N\hat{P}(\boldsymbol{n},\boldsymbol{o})}\sum_{k=1}^{N}\mathbf{1}_A(\boldsymbol{x}^k)H\left(\lambda(\boldsymbol{x}^k)-\hat{P}(\boldsymbol{n},\boldsymbol{o})\right), \tag{20}$$

where H is the same as in Equation 8.

4.3. Three-Scale Filtering

To further speed up rendering, we designed a hierarchical version of our algorithm, with three scales: micro-scale, meso-scale and macro-scale (see Figure 3). We pick the relevant scale depending on the number of particles in the pixel footprint:

- for a small number of particles (less than 4), we use the full model, *micro-scale*, as expressed in Equation 20: we traverse the spatial hierarchy to find particles located inside the pixel footprint and for each particle we use the DPF to decide the acceptance of the particle.
- for a medium number of particles (between 4 and 64), we use an approximate model, *meso-scale*: we do not check for individual acceptance for each particle and simply use Equation 18.
- for an even larger number of particles (more than 64), we use a stronger approximation, *macro-scale*: we do not conduct the spatial traversal and approximate the number of particles using the footprint area. We obtain the approximate particle count by multiplying the footprint area, the DPF and the total particle count.

Figure 4 visualizes these three scales, along with results from each scale and their combination. The companion video shows the transitions from micro-scale to macro-scale when the object is moving closer to the camera. The video confirms our assumption that the density of glints is a good criterion to decide the scale. We do not observe any visible discontinuity between these scales.

4.4. Pipeline

Our rendering pipeline can be subdivided into four main steps, with a total of seven passes:

- Precomputation:
  – Directional probability function (DPF) generation for varying roughness on the CPU.
- Update on Demand:
  – DPF buffer extraction and accumulation pass.
  – Environment map prefilter pass.
  – Spatial tree construction pass.
- Run-time:
  – Scale aware glint computation.
  – Directional occlusion computation.
- Postprocess:
  – Gaussian blurring for directional occlusion and glints.

The first pass (precomputation) is run on the CPU and can be reused for all isotropic BRDF models with Beckmann normal distribution function. The other passes run on the GPU. Only a subset of these passes is required for each frame, depending on the application: if the material is unchanged, only the last two steps (run-time and postprocess). If the material has been edited, some of the other passes need to be computed again, depending on the parameters changed.

We provide details for each pass as follows:

DPF Buffers Generation. We sample the surface roughness with $0.02, 0.04, 0.1, 0.2, 0.4, 0.6, 0.8, 1.0$. For each sampled surface roughness, we compute the directional probability function (DPF). We set the glint reflection angle γ as 1 degree. We use the same method as in Wang et al. [WWH18]. We store the DPF using eight 3D buffers (one for each parameter value) and transfer these buffers onto GPU. The resolution of each buffer is shown in Table 2.

DPF Extraction and Accumulation. Given the surface roughness, we extract the DPF from the DPF buffers by linear interpolation. We also do the accumulation for this table, considering the given γ.

Environment Map Prefiltering. We sample the surface orientation and the outgoing direction, considering the surface roughness. The sample count is shown in Table 2. For each sample, we preconvolve the BRDF and the incoming light, resulting in a Glint Buffer. We also compute the average DPF, resulting in a Filtered DPF Buffer.

Spatial Tree Construction. We generate the spatial hierarchy for the glints and cache it in a buffer. We build the hierarchy by multinomial splitting of particle count into child nodes, as in [JHY*14]. The details of the node structure is shown in Section 5. We did not generate it procedurally, like [JHY*14] and [WWH18], for two reasons: first, procedural generation is not efficient on the GPU; second, with the separable model [WWH18], the depth of the hierarchy is much smaller than the original model, thus the memory cost is affordable (see Table 2). We use a 1D buffer to store the quadtree.

Glint Rendering. We use the area of the pixel footprint to estimate the number of particles located in the footprint. With this number, we use the three-scale model Section 4.2 to get the radiance.

Directional Occlusion. We shoot a ray in the specular reflection of the view direction with respect to the surface normal, and record the visibility (0 or 1) of this ray into the fourth channel of the radiance buffer.

Gaussian Blurring. First, we blur the directional occlusion result (the fourth channel of the radiance buffer) with a Gaussian kernel of size 9, resulting in a filtered weight δ to approximate soft shadows from the environment map. The kernel size is a compromise, as a smaller kernel size fails to produce the smoothness, while a larger kernel size requires more computations. We then multiply the glint radiance computed in the previous step with δ. Finally, we blur this result with a Gaussian kernel of size 3 to avoid aliasings of glint shapes, which is also a compromise, as a larger value provides glints that are too blurry and a smaller value may lead to aliasing.

4.5. Applications

Our method allows for fast rendering while changing the view point, incoming light and material parameters. When material parameters are fixed, we only need to compute the last two steps (rendering and post-processing).

The main parameters for the glint material are: glint color, glint count N, search angle γ and surface roughness α. Editing the glint color does not require extra passes. The flake count N affects the spatial tree construction pass. It costs an extra 6 ms. The DPF Filtering pass (about 2 to 3 ms) and environment map prefilter pass (about 8 ms) have to be performed when the search angle γ or the surface roughness are modified (see Table 3).

5. Implementation Details

Point or directional light. We do not neet prefiltering with point or directional lights since the incoming light direction is fixed. Equation 13 becomes:

$$L_o(A, \boldsymbol{o}) = L_i(\boldsymbol{i}) \hat{D}'(A, \boldsymbol{o}, \boldsymbol{i}) M(\boldsymbol{i}, \boldsymbol{o}), \tag{21}$$

where $\hat{D}'(A, \boldsymbol{o}, \boldsymbol{i})$ is computed with Equation 5. $P(\boldsymbol{i}, \boldsymbol{o}, \gamma)$ is represented with a thee dimension table in our implementation, similar to Wang et al. [WWH18], except with a GPU buffer. We use our three-scale model, as with environment mapping, but using $P(\boldsymbol{i}, \boldsymbol{o}, \gamma)$ for particle acceptance rather than $\hat{P}(\boldsymbol{n}, \boldsymbol{o})$.

Other materials: To render diffuse materials under an environment map, we also use a prefiltered version of the environment map. Computations are easier, as only two dimensions for the normal orientation are required.

Two-step traversal: We traverse the spatial hierarchy, until we reach a node at a preset depth. We count the flakes located inside the footprint. We separate the traversal and the gathering operations by caching the nodes in a tree cut.

Node structure: The node structure uses two `float4`. The first `float4` includes the node ID, particle count, depth and child count; the second `float4` includes the min and max of the Axis Aligned Bounding Box (AABB) of the area. Although some data (e.g. depth, node ID, and AABB) can be retrieved implicitly during traversal, we store them for performance.

6. Results and Discussion

We have implemented our algorithm inside Mitsuba Renderer [Jak10]. We compared our algorithm against: (i) Jakob et al. [JHY*14] which we consider as the reference for quality validation,

(ii) Wang et al. [WWH18] and (iii) Zirr and Kaplanyan [ZK16a]. Both Jakob et al. [JHY*14] and Wang et al. [WWH18] are running on the CPU and used for quality validation. We used the implementation of Zirr and Kaplanyan [ZK16a] from [ZK16b] on the GPUas a baseline for performance comparison.

All timings in this section are measured on a 2.20GHz Intel(R) Xeon(R) CPU E5-2630 (40 cores) with 32 GB of main memory and NVIDIA GeForce RTX 2080 Ti GPU for real-time rendering using Optix (Cuda). Unless otherwise stated, we use the Beckmann microfacet model as the underlying smooth distribution D, which has a single roughness parameter with lower values corresponding to smoother surfaces. We only consider direct illumination.

6.1. Qualitative Validation

We first compare our method with the reference solution, Stochastic Microfacet Distribution, by Jakob et al. [JHY*14] in Figures 1, 5 and 6. Qualitatively, our approach produces results that are very similar to the reference. In terms of computation time, our method costs less than 30 ms for entire rendering process and only 10 ms for glint evaluation at Full HD resolution.

In Table 1, we show the average energy of the glints (total energy divided by glint count) for both our method and [JHY*14]. We found our method provides very similar average energy as the reference, which confirms the high quality of our rendered results.

6.2. Comparison with Real-time Glint Methods

In Figures 7 and 8, we compare our method with Zirr and Kaplanyan [ZK16a]. Figure 8 uses environment map illumination. Though their method is slightly faster than ours, our method provides higher quality. Our result captures the incoming light distribution from the environment map (shiny yellow on the left side of the right dress), while Zirr and Kaplanyan [ZK16a] method does not sample the environment map and is simply using a different light cone to fake the surface roughness. In Figure 7 illumination comes from a point light source. Our method produces results that are similar to the reference, while Zirr and Kaplanyan [ZK16a] method is visibly different. In the close-up view, we can spot visual artifacts.

6.3. Performance and Timings

Table 1 displays the settings for the materials and timings for all our test scenes. We report the computation time without glints, and the computation time for the reference method by Jakob et al. [JHY*14], Wang et al. [WWH18] and our method. We also provide the cost associated with glint computation. In all the test scenes, the entire rendering cost of our method is less than 30 ms. Among these cost, only less than 10 ms is spent on glint evaluation. The precomputation time of the first stage (DPF Generation) on CPU is 18 s. This step is independent of material properties or scenes, thus it can be reused.

Table 2 shows the cost of the precomputed buffers, including the Glint Buffer, DPF buffer and quadtree. The glint buffer cost depends on the surface roughness. A smoother surface results in less cost. We use uniform cost for the DPF, as we tabulated the 4D function using a uniform grid, and this is the same as in Wang et al. [WWH18]. We use a fixed depth, 11, for the spatial quadtree. This quadtree is only used for space subdivision, so the maximum depth is deep enough, and we find it works well for all the test scenes.

Figure 9 displays the costs for the individual components of our algorithm, using Optix ray-tracing. We visualize the cost of Ray tracing, Diffuse Shading, Directional Occlusion, Gaussian Filtering and Glint Evaluation. In all of our test scenes, the glint evaluation costs less than 50 % of the total cost. Ray-tracing is most expensive component, and strongly depends on image resolution and geometry complexity. Gaussian filtering component is also expensive, as it includes a filtering for directional occlusion with a relative large kernel size (9) and a filtering for glint shapes; it also depends on the image resolution. Directional occlusion is relatively cheap, as it requires only one ray per sample for partial occluded shading points.

Table 3 reports the cost of individual steps during glint material editing, including quadtree updating, Prefiltering and Gamma Accumulation for DPF. Depending on the editing parameters, only a subset of these passes are required. For example, changing the number of particles only requires computing the quadtree pass. Changing roughness and gamma requires computing the prefiltering and Gamma Accumulation passes. The prefiltering step is fast, as the tabulated resolution is low ($31 \times 60 \times 10 \times 36$). The time to edit the material is roughly the same as the time to render a frame.

In Figure 10, we show the impact of image resolution on rendering time over the Shoe Scene. The rendering cost is linear with respect to the number of pixels before reaching resolution 1280×720 and then becomes sub-linear after this point, benefiting from the multi-scale model.

6.4. Parameter Analysis

Figures 11 shows the impact of glint parameters (surface roughness) on material appearance with algorithm. For all parameters values, our method provides results that are similar to the reference. We found that computation time remains almost the same for all values of the parameters.

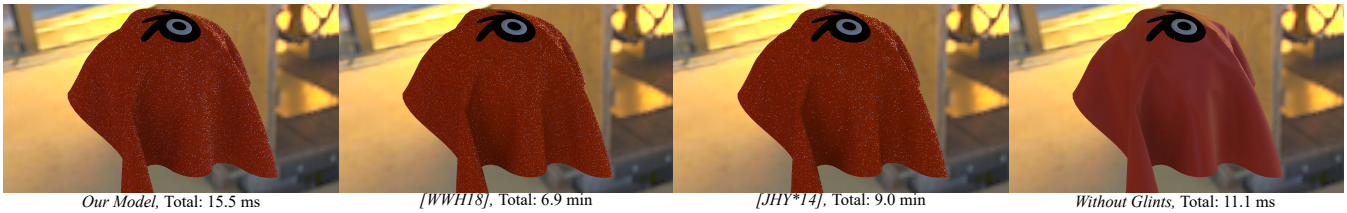
6.5. Discussion and Limitation

Our algorithm only considers point lights, directional lights and environment map. Other kind of lights, such as area lights, are left as an avenue for future work. The prefiltering method we used for environment maps can not be used for area lights, as different positions with the same orientation can have different incoming illumination. In Figure 12, we approximate the area light with an environment map. The difference is obvious, compared to the reference.

Our algorithm precomputes data for prefiltering or spatial hierarchy, avoiding heavy computations during rendering, at the expense of memory (see Table 2). The quadtree itself requires 256 MB, and the other buffers cost up to 16 MB. We used a naive representation for the prefiltered glint radiance, so a more compact representation would reduce this cost; existing works on compressed prefiltered

Name	Res.	α	γ	N	#Sam.	Glint Ev. Time	Total Time			Average Energy	
						Ours (ms)	Ours (ms)	[JHY*14] (m.)	[WWH18] (m.)	Ours	[JHY*14]
Shoe	1920 × 1080	0.1	6	10^6	1024	8.4	18.9	12.2	7.5	0.18	0.14
Snow	1920 × 1080	0.5	2	10^5	1024	9.3	26.4	20.6	15.5	11.4	11.7
Cloth	1920 × 1080	0.5	2	10^5	1024	4.4	15.5	9.0	6.9	0.069	0.074

Table 1: *Parameters and costs for the scenes used in this paper. Res. is the resolution of the rendered image. α is the roughness of the materials. γ is the search angle for the directional domain. N is the total particle count. #Sam. is the count of samples used during rendering for [JHY*14] and [WWH18]. Glint Ev. Time means the glint evaluation time in our method.*



Our Model, Total: 15.5 ms *[WWH18],* Total: 6.9 min *[JHY*14],* Total: 9.0 min *Without Glints,* Total: 11.1 ms

Figure 5: *Quality comparison with [JHY*14] and [WWH18] on the Table Cloth Scene. Glint Material: α: 0.5, N: 100K, γ: 2.0.*

Name	Glint Buffer		DPF Buffer	Quadtree
	4D Resolution	MB	MB	MB
Shoe	31 × 60 × 10 × 36	10.22	5.7	256
Snow	10 × 18 × 4 × 12	0.14	5.7	256
Cloth	10 × 18 × 4 × 12	0.14	5.7	256
Dress	Two Materials	10.36	5.7	256

Table 2: *Parameters and costs for precomputed data or buffers: glint buffer, DPF buffer (with Resolution 90 × 90 × 180) and quadtree. The maximum depth of quadtree is 11. 4D resolution corresponds to the lookup table resolution, with the first two dimensions for the normal orientation and the last two dimensions for the view direction.*

Name	Quadtree ms	Prefilter ms	Gamma Acc. ms	Total ms
Cost	6.4	8.0	2.7	17.1
Glint Property		√	√	10.7
Glint Number	√			6.4

Table 3: *Top row: costs for each pass during glint editing; second row: costs for glint properties (reflection angle γ and surface roughness α) editing; third row: costs for glint number editing. These costs are for the Shoe Scene.*

environment maps, such as Kautz et al. [KVHS00] and Ramamoorthi et al. [RH02], could be used in combination with our work.

We build the hierarchy in a individual pass rather than building it procedurally in the rendering pass in our implementation for performance reasons, as building a hierarchy on the fly is not suitable on the GPU. The disadvantage is that the storage is limited by the glint number. A future improvement would be to switch automatically between precomputed hierarchy and building it on-the-fly, depending on the glint number.

7. Conclusion

We have presented a new method for real-time rendering of glints. Our method is based on a prefilterable stochastic microfacet model and a scale aware spatial traversal model. We provide a full GPU implementation with 7 passes. Our method allows for very fast rendering of scenes with glints, with fully dynamic geometry, illumination and material properties. Without editing of material properties, our method costs only 30 ms for the entire rendering, and 10 ms for glint evaluations using FHD resolution. With edition, it costs at most an extra 10 ms, depending on the parameters being edited. We believe our algorithm will be a useful tool for real-time rendering of glints and sparkles in game or movie development industry.

In future works, we want to extend our approach to other material properties such as scratches. We also plan to improve the representation model of the prefiltered glint radiance. We want to improve the performance by combining our algorithm with Zirr and Kaplanyan [ZK16a] by sampling the binomial distribution per tree node. We also want to reduce the number of samples by importance sampling the DPF during prefiltering.

Acknowledgements. We thank the reviewers for the valuable comments. This work has been partially supported by the National Key R&D Program of China under grant No. 2017YFB0203000, the National Natural Science Foundation of China under grant No. 61802187 and 61872223, the Natural Science Foundation of Jiangsu under grant No. BK20170857.

References

[AK16a] ATANASOV A., KOYLAZOV V.: A practical stochastic algorithm for rendering mirror-like flakes. In *ACM SIGGRAPH 2016 Talks* (2016), pp. 67:1–67:2. 2

[AK16b] ATANASOV A., KOYLAZOV V.: A practical stochastic algorithm for rendering mirror-like flakes. In *ACM SIGGRAPH 2016 Talks* (New York, NY, USA, 2016), Association for Computing Machinery. 2

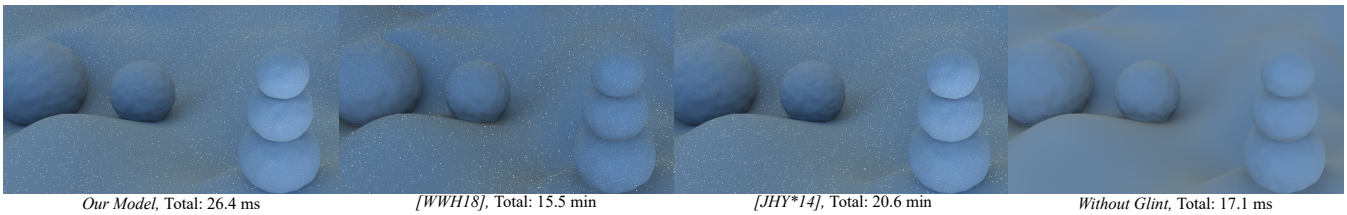


Figure 6: *Quality comparison with [JHY*14] and [WWH18] on the Snow Scene. Glint Material: α 0.5, N: 100K, γ: 2.0.*

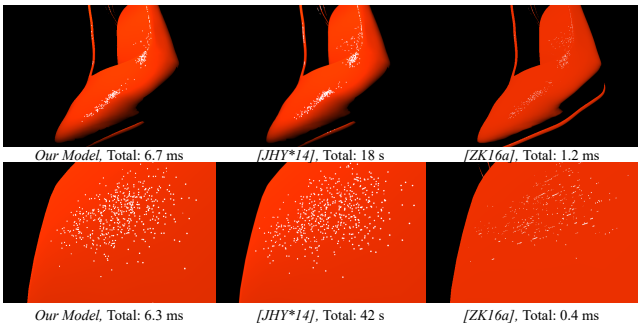


Figure 7: *Comparison with reference [JHY*14] and [ZK16a] on the Shoe Scene with point light only. Glint Materials: roughness: 0.1, N: 1M, γ: 6.0. Resolution:* 768×512.

[BW15] BOWLES H., WANG B.: Sparkly but not too sparkly! anti-aliasing a procedural sparkle effect. In *Siggraph 2015 Advances in Real-time Rendering in Games* (Los Angeles, United States, Aug. 2015). 2

[CT82] COOK R. L., TORRANCE K. E.: A reflectance model for computer graphics. *ACM Trans. Graph. 1*, 1 (Jan. 1982), 7–24. 1, 2, 3

[GGN18] GAMBOA L. E., GUERTIN J.-P., NOWROUZEZAHRAI D.: Scalable appearance filtering for complex lighting effects. *ACM Trans. Graph. 37*, 6 (2018). 2

[Jak10] JAKOB W.: Mitsuba renderer. http://www.mitsuba-renderer.org/, 2010. 6

[JHY*14] JAKOB W., HAŠAN M., YAN L.-Q., LAWRENCE J., RAMAMOORTHI R., MARSCHNER S.: Discrete stochastic microfacet models. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2014) 33*, 4 (2014). 1, 2, 3, 4, 6, 7, 8, 9, 10

[KC08] KŘIVÁNEK J., COLBERT M.: Real-time shading with filtered importance sampling. *Computer Graphics Forum 27*, 4 (2008), 1147–1154. Eurographics Symposium on Rendering, EGSR '08. 3

[KVHS00] KAUTZ J., VÁZQUEZ P.-P., HEIDRICH W., SEIDEL H.-P.: Unified approach to prefiltered environment maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 185–196. 2, 3, 7

[MLH02] MCALLISTER D. K., LASTRA A., HEIDRICH W.: Efficient rendering of spatial bi-directional reflectance distribution functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2002), HWWS '02, pp. 79–88. 3

[RH02] RAMAMOORTHI R., HANRAHAN P.: Frequency space environment map rendering. *ACM Trans. Graph. 21*, 3 (July 2002), 517–526. 7

[Sho12] SHOPF J.: Gettin' procedural. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Shopf-Procedural.pdf, 2012. 2

[WB16] WANG B., BOWLES H.: A robust and flexible real-time sparkle effect. In *EGSR 2016 E&I - Eurographics Symposium on Rendering - Experimental Ideas & Implementations* (Dublin, Ireland, 2016), The Eurographics Association, pp. 49–54. 2

[WMLT07] WALTER B., MARSCHNER S. R., LI H., TORRANCE K. E.: Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (2007), EGSR'07, pp. 195–206. 1, 2, 3

[WRG*09] WANG J., REN P., GONG M., SNYDER J., GUO B.: All-frequency rendering of dynamic, spatially-varying reflectance. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 133:1–133:10. 3

[WWH18] WANG B., WANG L., HOLZSCHUCH N.: Fast global illumination with discrete stochastic microfacets using a filterable model. *Computer Graphics Forum (Proceedings of Pacific Graphics 2018) 37*, 7 (2018), 55–64. 1, 2, 3, 4, 6, 7, 8, 9

[YHJ*14] YAN L.-Q., HAŠAN M., JAKOB W., LAWRENCE J., MARSCHNER S., RAMAMOORTHI R.: Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2014) 33*, 4 (2014). 2

[YHMR16] YAN L.-Q., HAŠAN M., MARSCHNER S., RAMAMOORTHI R.: Position-normal distributions for efficient rendering of specular microstructure. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2016) 35*, 4 (2016). 2

[ZK16a] ZIRR T., KAPLANYAN A. S.: Real-time rendering of procedural multiscale materials. In *Proceedings of I3D '16* (2016), I3D '16, pp. 139–148. 2, 7, 8, 9, 10

[ZK16b] ZIRR T., KAPLANYAN A. S.: Real-time rendering of procedural multiscale materials. https://www.shadertoy.com/view/ldVGRh, 2016. 7

Figure 8: *Comparison with reference [JHY*14] and [ZK16a] on the Dress Scene. Glint Materials for the dress on the left: roughness: 0.1, N: 1M, γ: 6.0; the dress on the right: α: 0.5, N: 100K, γ: 2.0.*

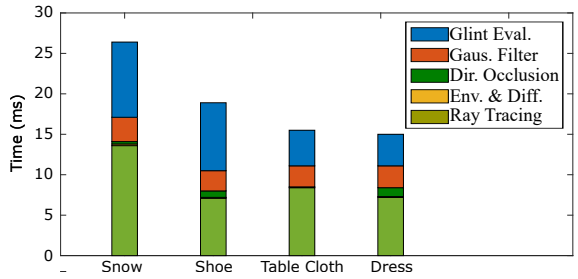


Figure 9: *Computation time for each component (Ray tracing, Diffuse Shading, Directional Occlusion, Gaussian Filtering and Glint Evaluation) in our algorithm for four scenes.*

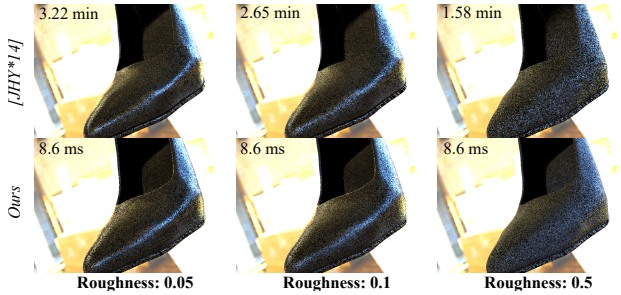


Figure 11: *Comparison between our method and Jakob et al. [JHY*14] with varying surface roughness for the Shoe Scene. Resolution:* 768 × 512.

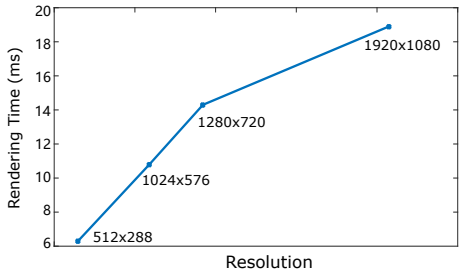


Figure 10: *Rendering time as a function of image resolution (number of pixels) on the Shoe Scene.*

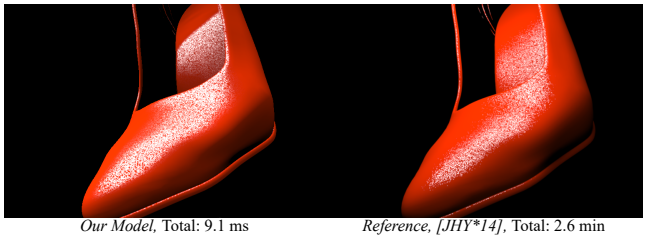


Figure 12: *Comparison between our method, and Jakob et al. [JHY*14] with an area light source for the Shoe Scene. We approximate the area light with an environment map in our method. Resolution:* 768 × 512.